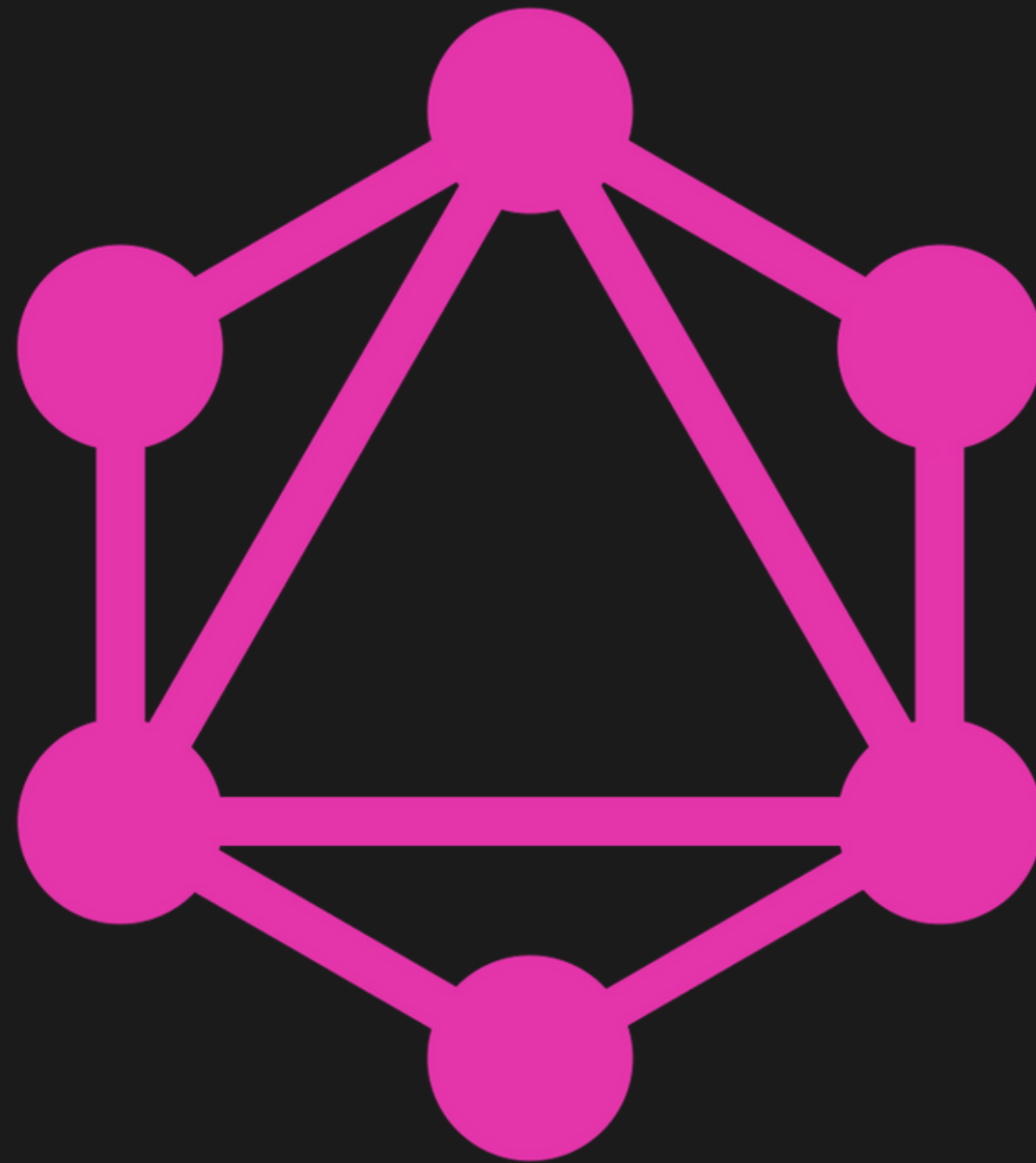
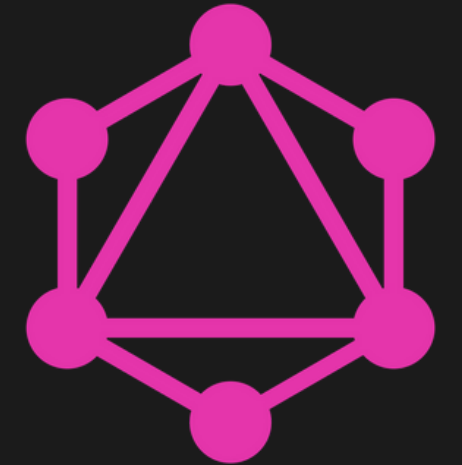


GraphQL



Che cos'è GraphQL?



- Creato da Facebook nel 2012
- GraphQL è un'estensione di REST
- GraphQL usa un Query Language
- Recupera i dati in maniera puntuale

GraphQL ~~vs~~ REST

- Un solo endpoint per l'accesso ai dati
- No Under-fetching / Over-fetching
- Basso utilizzo della banda
- Dati Fortemente Tipizzati
- No Versioning
- Mock del backend

GraphQL REST

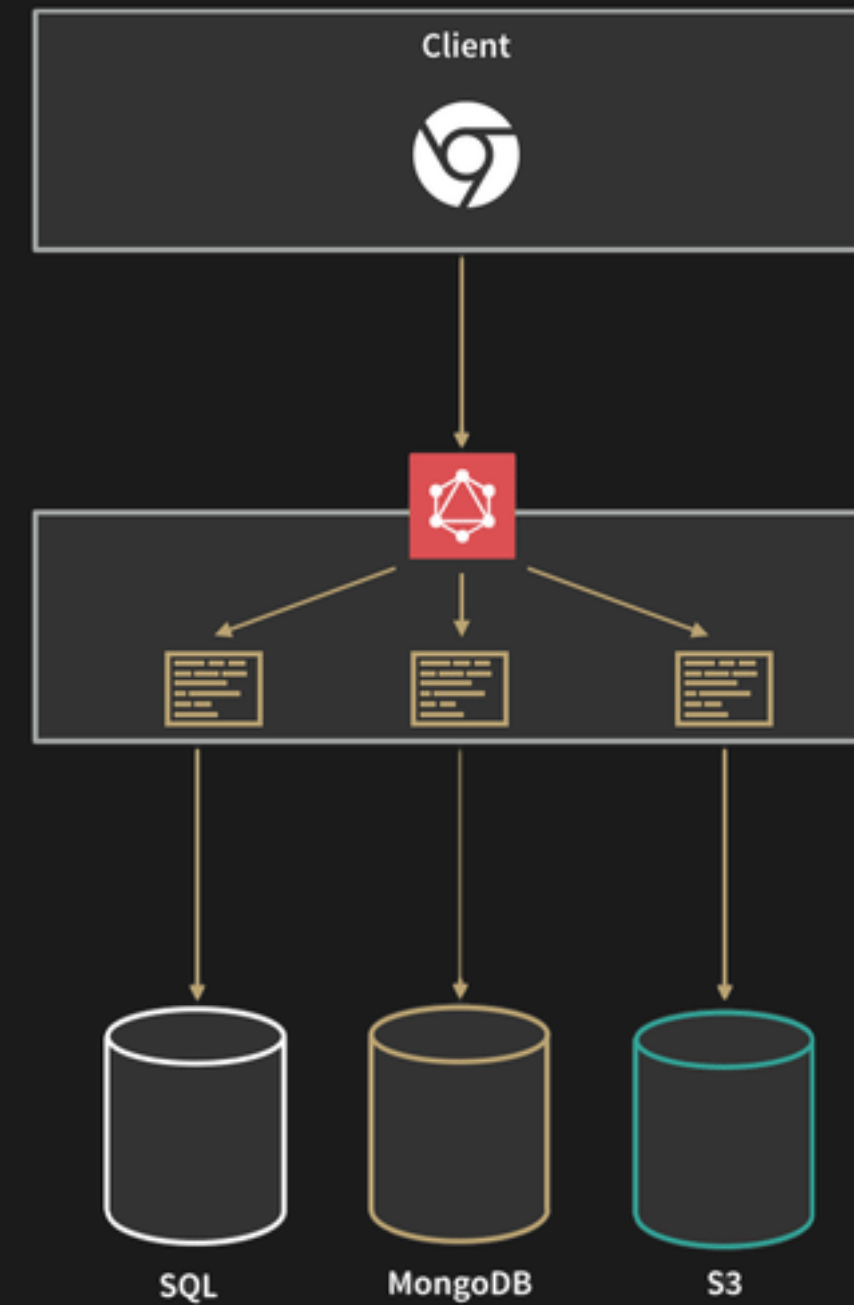
- Caching complesso
- Più difficile da ottimizzare
- Non adatto per la restituzione di File


GraphQL vs REST

R E S T

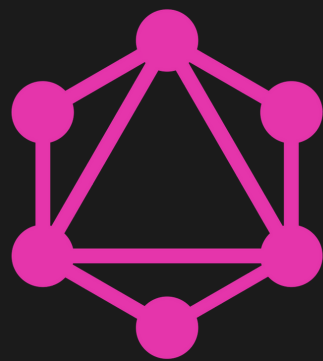


G R A P H Q L



 = Arbitrary code

Aspetti Principali



SCHEMA DEFINITION (SDL)

Descrive la composizione dei dati con uno schema

QUERY

Il client richiede i dati in maniera puntuale

MUTATION

Il client può aggiungere/modificare/cancellare i dati

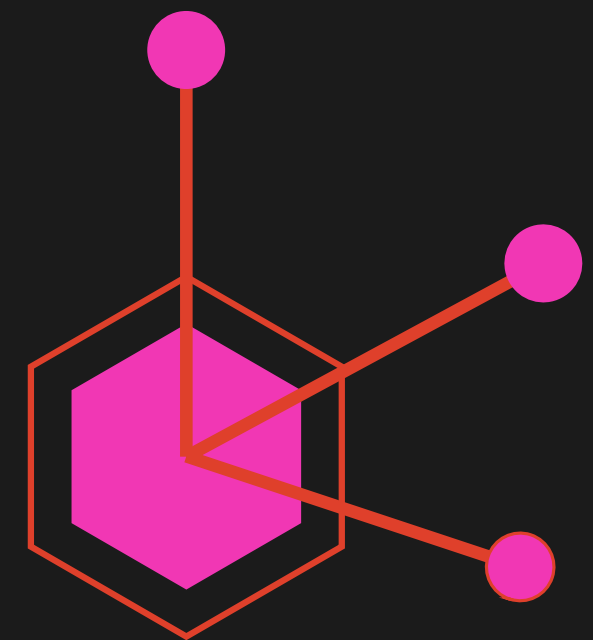
SUBSCRIPTION

Client in ascolto e ogni evento aggiorna in real time UI

Schema Definition Language

```
type Author {  
  id: Int!  
  firstName: String  
  lastName: String  
  posts: [Post]  
}  
  
type Query {  
  posts: [Post]  
  author(id: ID!): Author  
}  
  
type Post {  
  id: Int!  
  title: String  
  author: Author  
  votes: Int  
}
```

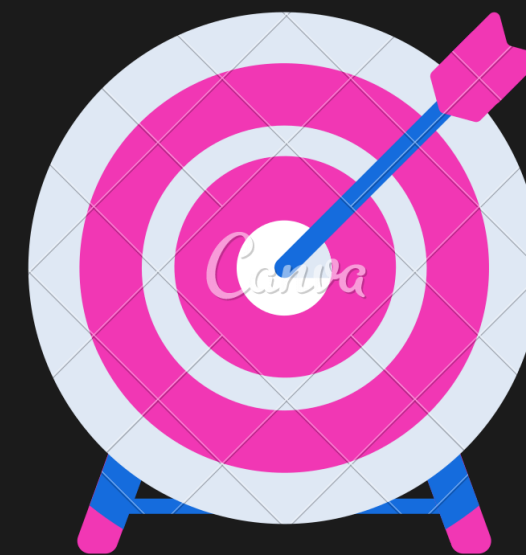
- Definizione di richieste e azioni che è possibile compiere da client
- Struttura dati fortemente tipizzata
- Possiamo avere dati primitivi (int, string, bool) oppure un oggetto



Query

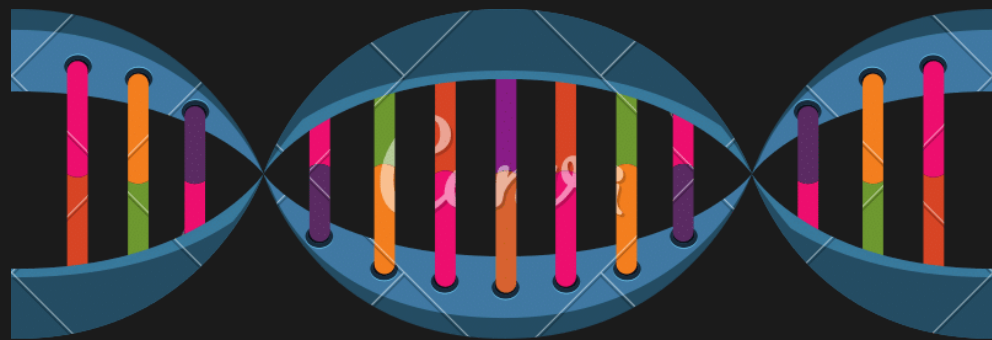
```
1 query PostsForAuthor {  
2   author(id: "1") {  
3     firstName  
4     posts {  
5       title  
6       votes  
7     }  
8   }  
9 }  
10  
11
```

Il client può richiedere il campo o i campi di cui ha bisogno invece di ottenere i dati per tutti i campi da un particolare tipo. Ovviamente è possibile utilizzare solo i campi che sono stati esposti dall'API



Mutation

Le mutations vengono utilizzate per inviare dati al server, ovvero è possibile aggiungere, modificare o cancellare i dati. Il client può solo sfruttare le mutations che sono state esposte dallo schema per modificare i dati.



```
POST https://localhost:5001/graphql/

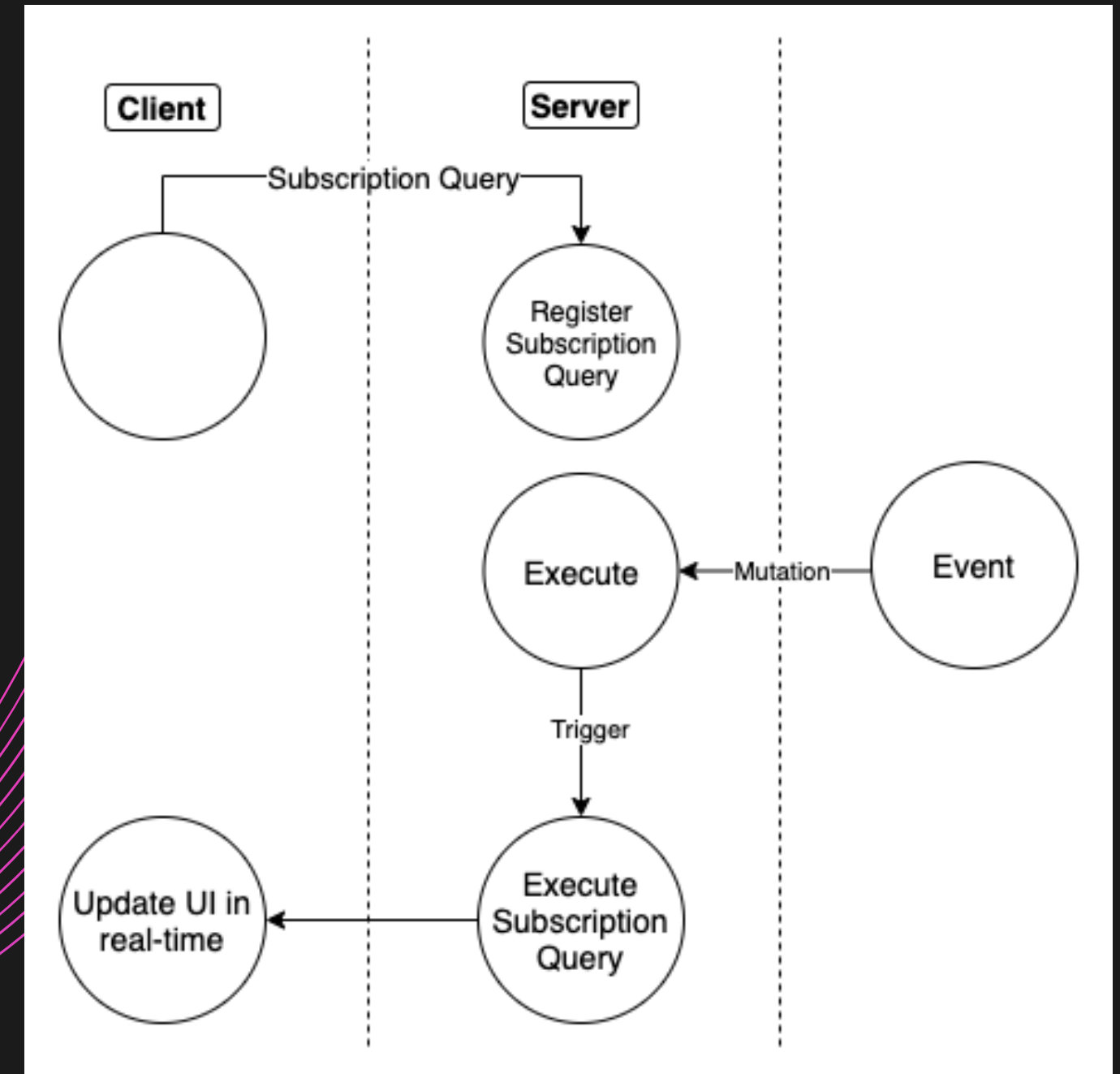
{
  "inputType": {
    "ProductName": "Think Pad",
    "Brand": "Samsung",
    "Cost": 7500,
    "Type": "Laptop"
  }
}

mutation ($inputType: GadgetInput!) {
  save(input: $inputType) {
    Id
  }
}
```

```
{
  "data": {
    "save": {
      "Id": 4002
    }
  }
}
```

Subscription

Le subscriptions consentono a un server di inviare dati ai propri client, avvisandoli quando si verificano eventi. Le sottoscrizioni forniscono supporto per architetture guidate da eventi e per le notifiche in tempo reale sfruttano WebSocket.



Subscription

Attualmente vengono supportati 2 tipi di subscriptions:

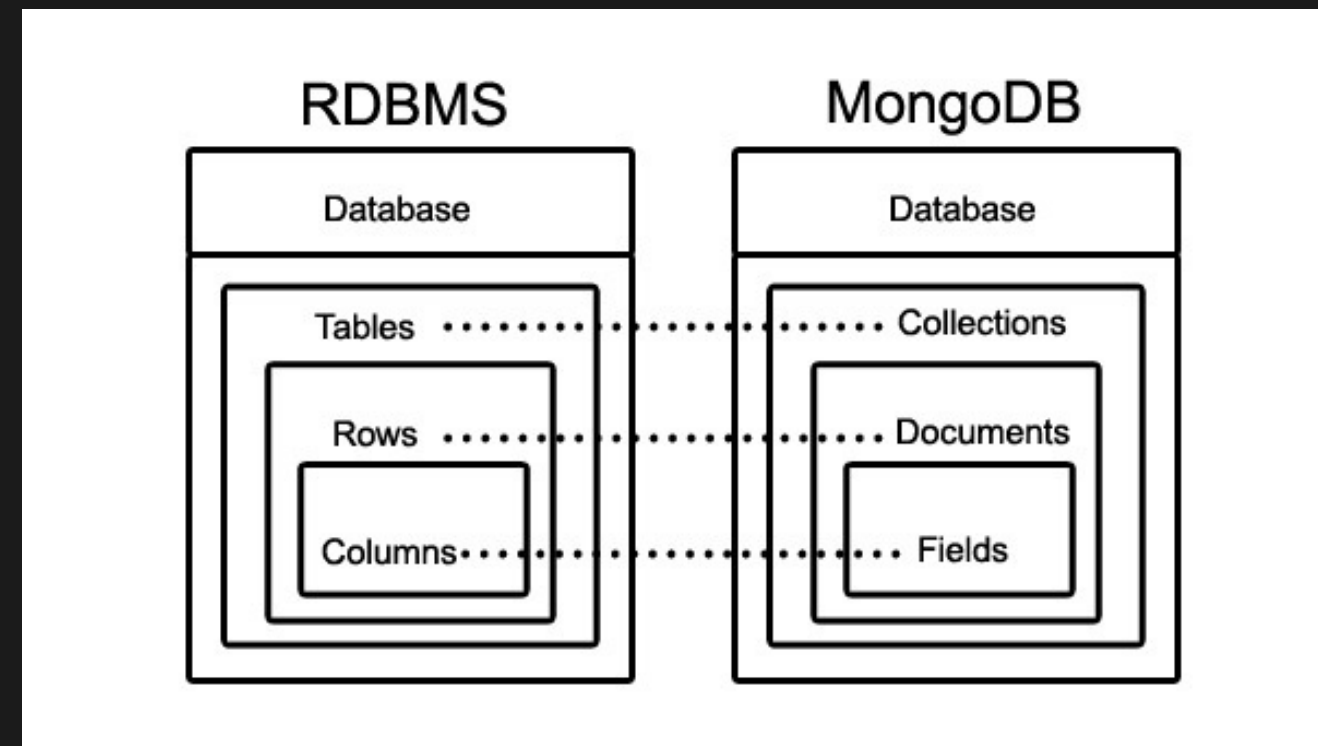
- InMemory - Utilizzato se abbiamo un singolo server e tutti gli eventi vengono attivati attraverso le nostre mutations.
- Middleware integrations - Un provider di abbonamento pronto all'uso che utilizza la funzionalità di pubblicazione/sottoscrizione. Se abbiamo più istanze del nostro server. (Redis <Message Broker> - HotChocolate)

Example React Client + Mock Server

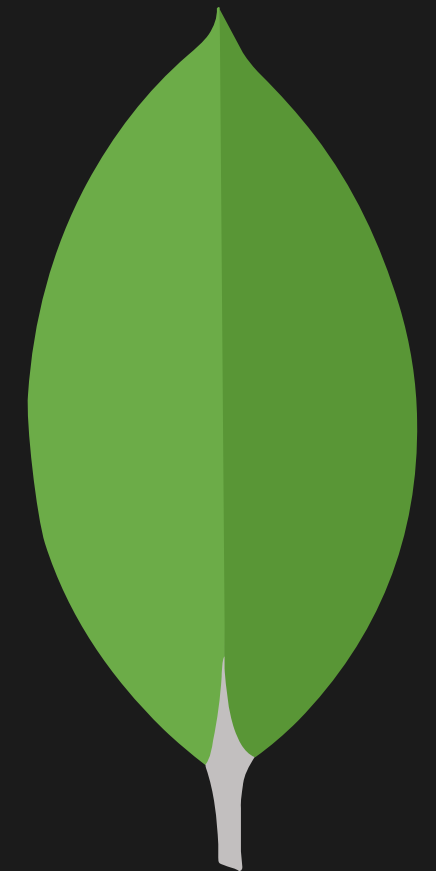
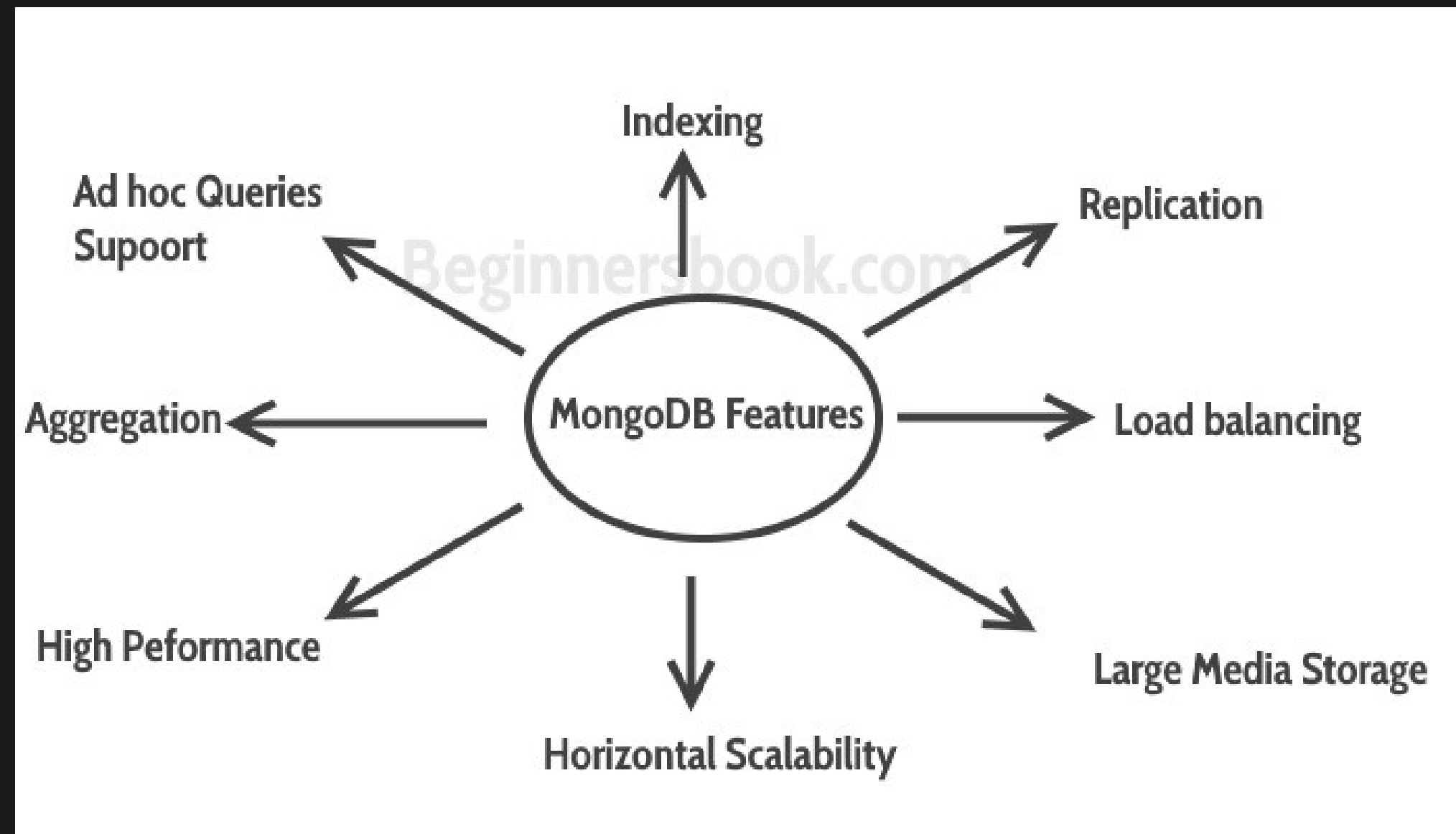




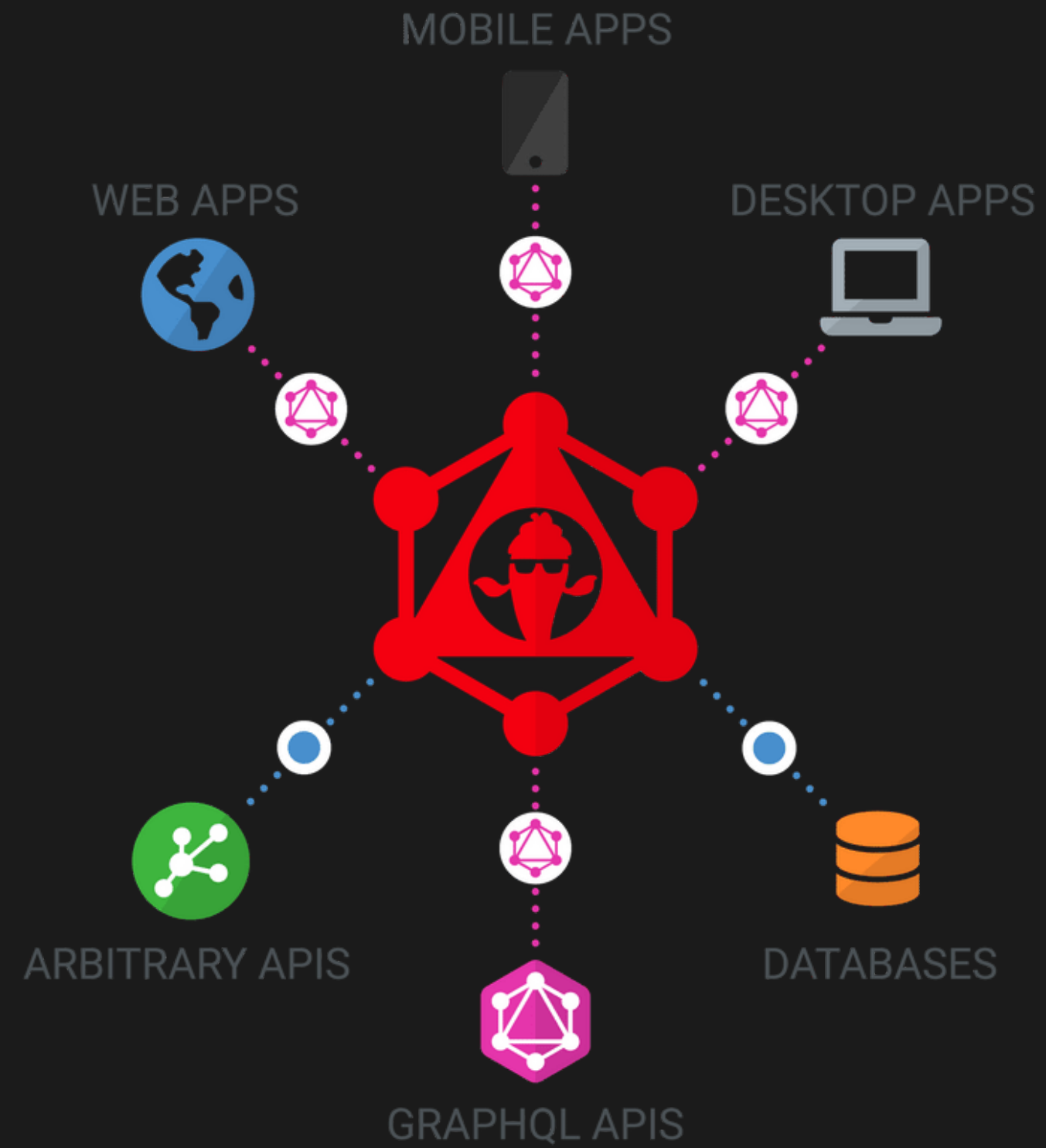
- Un database NOSQL open source orientato ai documenti
- Non usano uno SQL (Structured Query Language), il linguaggio classico dei database relazionali.
- scritto in C++, salva i dati nel formato BSON (Binary JSON), basato su JSON (JavaScript Object Notation).



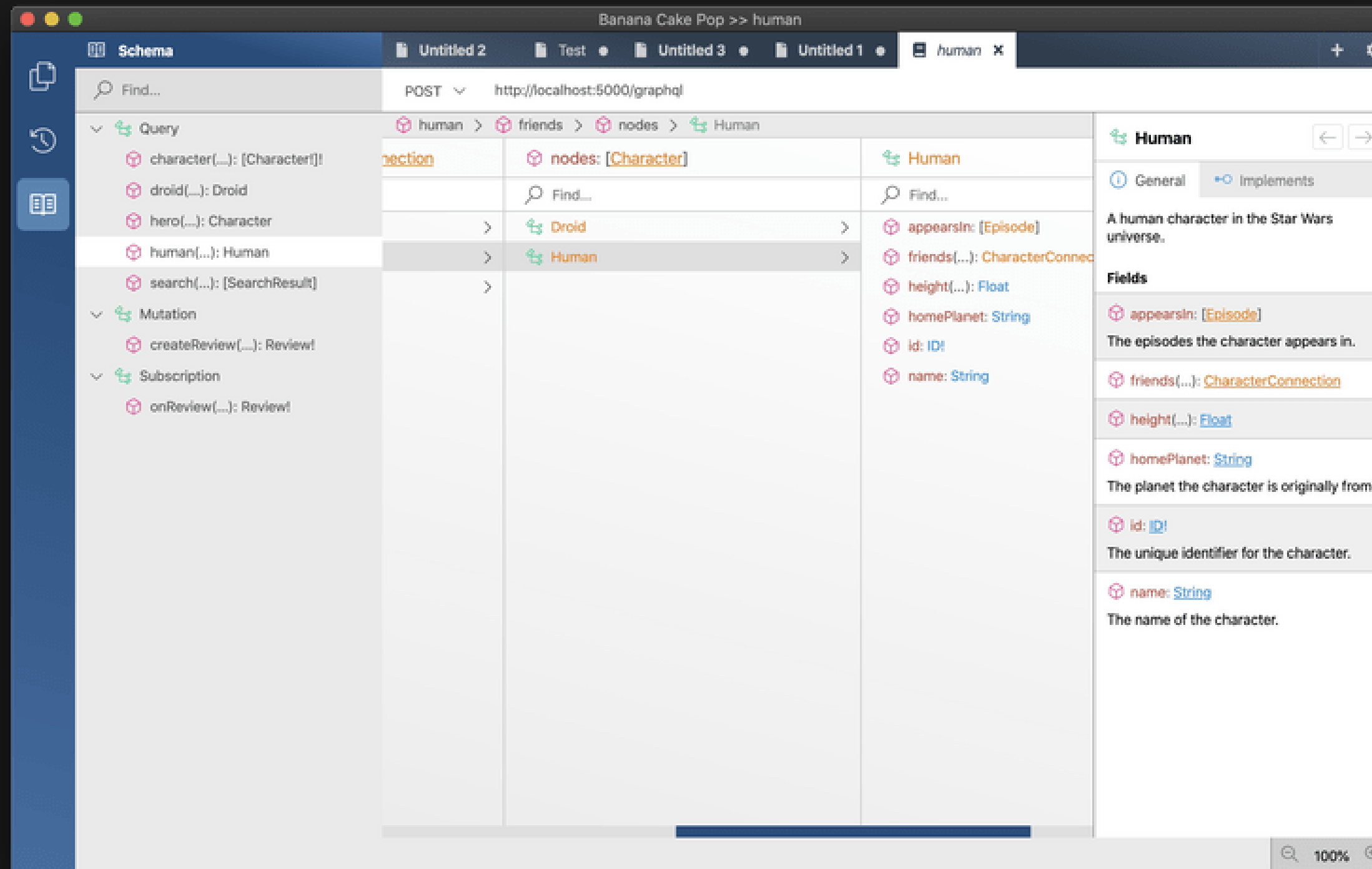
Perchè MongoDB?



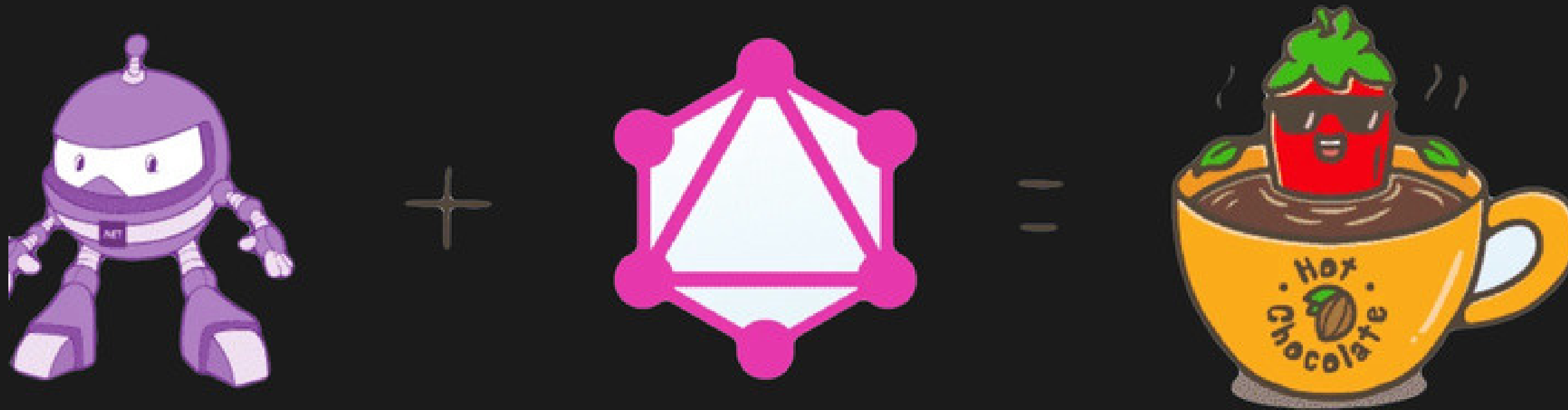
Hot Chocolate



Banana Cake Pop



Example Hot Chocolate + Mongo Db



https://github.com/Donato09/TrulliManager_MongoDB.git

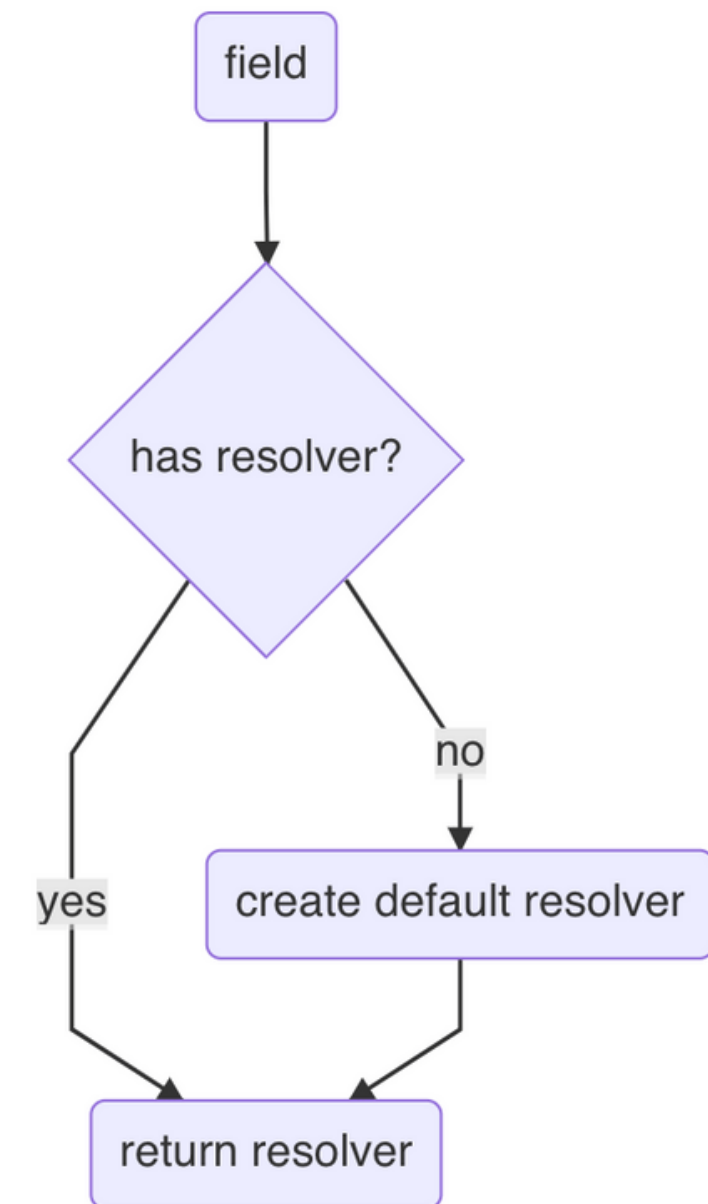
<https://github.com/Donato09/GraphQL-example.git>

Hot Chocolate Features

Resolvers:

Un resolver è una funzione generica che recupera i dati da un'origine dati arbitraria per un campo particolare.

Anche se non esiste un resolver definito per un campo, Hot Chocolate creerà un resolver predefinito per questo particolare campo dietro le quinte.



Hot Chocolate Features

Filters and Sorting:

Con i filtri Hot Chocolate è possibile esporre oggetti filtro complessi tramite l'API GraphQL che si traducono in query di database native. Oltre a questo Hot Chocolate permette anche di ordinare in maniera crescente o decrescente

Hot Chocolate Features

Projections:

Ogni richiesta GraphQL specifica esattamente quali dati devono essere restituiti. Il overfetching e underfetching insufficiente può essere ridotto o addirittura eliminato. Le proiezioni di HotChocolate sfruttano questo concetto e proiettano direttamente le query in arrivo nel database. Le proiezioni funzionano su IQueryable per impostazione predefinita,

Example Query Mutation & Subscription

```
query {  
  properties {  
    nodes {  
      id,  
      name,  
      trulli {  
        name  
      }  
    }  
  }  
}  
  
query {  
  trulli ( first: 2 ) {  
    nodes {  
      id,  
      name  
    }  
  }  
}  
  
subscription {  
  onTrulloGet {  
    name  
    propertyId  
  }  
}  
  
query {  
  trulloById (id: 3) {  
    name  
  }  
}
```

Example Query Mutation & Subscription

```
subscription {  
  onTrulloCreate {  
    name,  
    price  
    propertyId  
  }  
}
```

```
{ "inputType": {  
  "name": "Trullo Luxury",  
  "description": "Trullo più costoso",  
  "price": 200,  
  "capacity": 10,  
  "propertyId": 1  
}}  
mutation( $inputType: CreateTrulloInput  
) {  
  createTrullo(trullo: $inputType) {  
    id  
  }  
}
```

```
{  
  "inputType": {  
    "id" : 8  
  }  
}  
mutation( $inputType: DeleteTrulloInput  
) {  
  deleteTrullo( deletedTrullo:  
    $inputType) { id }  
}
```


THANK
YOU

